Implementation of a very large atmospheric correction lookup table for ASTER using a relational database management system

Alex Murray
Bjorn Eng
California Institute of Technology/Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099

Kurtis Thome
Optical Sciences Center, University of Arizona
Tucson, AZ 85721

June 20 1996

ABSTRACT

The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) is designed to provide a high resolution map of the Earth in both visible, near-infrared, and thermal spectral regions of the electromagnetic spectrum. The ASTER Science Team has developed several standard data product algorithms, but the most complex and computing-intensive of these is the estimation of surface radiance and reflectance values, which is done by modeling and correcting for the effects of the atmosphere. The algorithm for atmospheric correction in the visible bands sensed by ASTER calls for the use of a very large Atmospheric Correction Look Up Table (ACLUT). The ACLUT contains coefficients which describe atmospheric effects on ASTER data under various conditions. The parameters used to characterize the atmosphere and its effects on radiation in the ASTER bands include aerosol and molecular optical depth, aerosol size distribution, single scattering albedo, and Solar, nadir view, and azimuth angles. The ACLUT coefficients are produced by thousands of runs of a Radiative Transfer Code (RTC) program produced by Phil Slater and Kurt Thome of U. of A. The final version of the ACLUT is expected to be in the neighborhood of 10 Gigabytes. The RDBMS Sybase is used to manage the process of generating the ACLUT as well as to host the table and service queries on it. Queries on the table are made using ASTER band number and seven floating-point values as keys. The floating-point keys do not necessarily exactly match key values in the database, so the query involves a hierarchical closest-fit search. All aspects of table implementation are described.

Keywords: ASTER, atmospheric correction, sybase, relational data base, query, RDBMS, lookup

1. OVERVIEW OF ASTER

The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) will be one of the. instruments hosted on the EOS AM-1 platform, the first spacecraft of a series to be launched as part of NASA'S Earth Observing System (EO S). ASTER is being built in Japan under the auspices of the Ministry for International Trade and Industry (M ITI), in cooperat ion with NASA.

ASTER will be the only high spatial resolution imaging instrument on the AM-1 spacecraft. It will collect data in visible, shortwave infrared, and thermal infrared wavelengths, ASTER's data will contribute uniquely to several areas of investigation, including studies of land usage, surface composition and distribution of minerals, soil characteristics, effects of coastal erosion, changes in sea level, sea ice processes, and cloud characteristics. The thermal bands in ASTER will allow detailed studies of surface temperature, radiation balance, vegetation, soils, and evaporation and evapot ranspirat ion processes in addition to volcanic processes. In addit ion, ASTER has a backward-looking t elescope which provides a stereo band, enabling the production of high-resolut ion, high-qualit y Digital Elevation Models (DEMs).

The ASTER instrument consists of three separate subsystems or telescopes: the Visible and Near Infrared (VNIR) subsystem, the Shortwave Infrared (SW] R) subsystem, and the Thermal Infrared (TIR) subsystem. The characteristics of each of the telescopes are shown in Table 1.

Table 1 (from ref. [1]) - *Characteristics of A STER's telescopes*

CHARACTERISTIC	VNIR	SWIR	TIR
Spectral Bands (band number: range n microns)	1:0,52-0.60 2:0.63-0.69 3N: 0.76-0.86 3B: ..76- (3.86	4: 1.60 - 1.70 5: 2.145 -2.185 6:2.185- 2.225 7:2.235-2.285 8:2.295-2.365 9:2.360-2.430	10: 8.125 -8.475 11:8.475-8.825 12:8.925-9.275 13:10.25 - 10,95 14: 10.95- 11.65
Ground Resolution (m)	15	30	90
Data Rate (Mbit/second)	6'2	23	4.2
Cross-Track Pointing (degrees)	+/-24	+/-8.55	+/-8.55
Swath Width (km)	60	60	60
Quantization (bits)	8	8	12

Most Of the data products which will be generated from ASTER data depend heavily on accurately compelling, and correcting effects of the atmosphere on the data, There are two algorithms used for this purpose; one works on data from the TI R telescope, and the. other is used to process the data from the VNIR and S WIR telescopes. The At mospheric Correct ion Lookup Table (ACLUT) is employed in this latter algorithm, which is described in the following sect ions.

2. THE ATMOSPHERIC CORRECTION (VNIR/SWIR) ALGORITHM

The ASTER PGS implements the Atmospheric Correction algorithm described in Slater [4] and Thome [5]. Atmospheric data from external sources (other Instruments on AM-1, NMC or DAO) are used to derive a set of parameters describing the atmosphere at the time of ASTER data acquisition. MODTRAN, a well known atmospheric modeling program from the Air Force Geophysics Laboratory (see [6]), is used to correct for gaseous absorption. Aerosol effects are corrected using data generated by a Radiative Transfer Code (RTC) supplied by the Remote Sensing Group at the University of Arizona, described in the next section. Since the RTC is very computationally intensive, a lookup table approach is used. The relational database management system Sybase is being used to contain the results of a large number of RTC runs. This database is called the Atmospheric Correction Lookup Table (ACLUT). By querying the database with the derived atmospheric parameters, and parameters describing the viewing geometry, a table of RTC results is obtained. Linear interpolation into this table, using MODTRAN-corrected ASTER data, yields the corrected surface radiance and reflectance values.

The lookups and MODTRAN runs are performed for a subset of the locations within the scene. These locations are referred to as gridpoints for MODTRAN and grid-cells for queries in the ACLUT. The two grids are independent of one another and can have different resolutions. The density of gridpoints/cells is based on the resolution of the available external data sets. The MODTRAN grid will generally be selected to match the grid of the selected water-vapor profile data set; data from the Moderate Resolution imaging Spectroradiometer (MODIS) instrument is the preferred source. The ACLUT grid is based on one of the aerosol-related data sets, most likely from the Multi-Angle imaging Spectroradiometer (MISR) instrument. Typical grid sizes could be 15 km resolution for the MODTRAN grid and 5 km for the ACLUT grid,

For each gridpoint, MODTRAN is run a number of times to determine the transmittance (due to gaseous absorption) of the path from the Sun to the ground and up to the satellite. The MODTRAN results are convolved with ASTER band pass filter data to determine ASTER band transmittances. For each pixel, the MODTRAN outputs generated for the surrounding gridpoints are interpolated to compute the transmittance values used to correct the pixel.

Then for each ACLUT grid-cell, a query is made to the database. The results of that query are used in correcting all pixels in the grid-cell. Each table lookup returns an "answer" that consists of two columns of data, one for top-of-atmosphere radiance, and one for surface radiance. A third column, surface reflectance, is constant throughout the ACLUT since reflectance is an input to the RTC which generated the table. The MODTRAN-corrected top-of-atmosphere radiances are used to interpolate to the correct offset into the top-of-atmosphere radiance column yielded by the query. The same offset is used in the surface reflectance and surface radiance columns. A final adjustment to surface radiance is made using DEM slope data,

3. THE SOURCE OF THE DATA IN THE TABLE

The data contained in the ACLUT is generated by an implementation of a Radiative Transfer Code (RTC) algorithm. The RTC used here numerically solves the radiative transfer equation for a plane-parallel, horizontally-homogeneous atmosphere using Gauss-Seidel iteration [2]. The current version of the code yields radiances for any specified solar angle and satellite view direction. The surface is assumed to be horizontally homogeneous. We can allow the surface to be non-lambertian, but only the lambertian case is used to generate the ASTER ACLUT. Arbitrary aerosol scattering phase functions can be used but the look-up table version assumes Mie scattering. Data obtained from Iqbal [3] are used to convert relative radiances given by the code to absolute radiances.

The vertical distribution Of aerosols follows the 1 976 U.S. Standard atmosphere. Cirrus clouds are treated as aerosols distributed throughout the at mosphere instead Of in narrow layers. In the case Of a large volcanic eruption, the. look-up table will be regenerated using a vertical distribut ion based upon measured data. Regenerat ing the table will came delays in the processi ng of some data sets, but this is preferred to generating ext ra table cases which may never be necessary.

Atmospheric aerosol parameters are based upon optical depth and aerosol size distribution data obtained from MISR which is on the EOS-AM1 platform with ASTER. In all cases, an equivalent Junge size distribution is determined because the single parameter used to describe the distribution makes it attractive for use in a look-up table approach. Absorption by aerosols is treated in the table through the single-scatter albedo which will also be obtained from MISR data.

The RTC is monochromat ic, so band absorpt ion can onl y be handled approximately. A bsorption by gaseous constituents is found from a modified version of MODTRAN. Columnar amounts of absorbing gases will be obtained from MODIS which is also on the EOS-AM 1 platfor m. A lternate sources of data are being investigated in the. case of missing data from MODIS or MISR.

The RTC is implemented in Fortran. The program takes several inputs, including wave'lt-'n~th, Junge parameter (a, k, a, aerosol size distribut ion), real and imaginary indexes of refraction, and minima, maxima, and deltas for each of aerosol opt ical depth, molecular opt ical depth, nadir view angle, azimuth angle, and solar zenit h angle. The program computes a value of single scatter al bedo based on t he real and imaginary indexes of refract ion. The program's main outputs are atmospheric models, one for ever y combinat ion of the. input values which vary. Each mode] is a table of 11 rows and 3 columns. The first column contains reflectance values. This column is constant in all models - it contains the values 0, .1, .2, . . ., .9, 1,(J. The second column is surface radiance, and the third contains the corresponding at-satellite radiance. In corr ecting a radiance value observed by the instrument a mode] is used as follows: locate the observed val ue in the at -sate.lli(e radiance column, thus selecting a row in the table. The value in the sut'face radiance col umn is the ground radiance that would have produced the observed at-satellite radiance value given the atmospheric and geomet ric conditions used in generat ing the model. Simi lar1 y t he surface r eflectance value would have been the t r ue reflectance of the surface in order for the observed radiance to be produced by the surface radiance under the conditions assumed by the model. In practice, linear interpolation is used to calculate a row in the model which corresponds to the obser ve d at-satellite radiance value because the observed at-satellite radiance will rarely match a value in the model.

The RTC program loops on the variable quantities which are among the inputs. The program's action is perhaps best described as follows:

```
for each value of aerosol optical depth
    for each value of molecular optical depth
        for each value of solar zenith angle
            for each of the 11 surface reflectance values
                calculate a vector of surface radiance values
            end
            for each value of nadir view angle
```

```
                for each value of relative azimuth angle
                    calculate a vector of at-satellite radiance values
                    for each of the 11 surface reflectance values
                        calculate a vector of at-satellite radiances
                    end
                end
            end
        end
    end
end
```

For each combination of aerosol optical depth, molecular optical depth, and solar zenith angle, the program calculates a column of surface radiance values corresponding to the reflectance column. For each column of surface radiance values, the program calculates a column of at-satellite radiance values corresponding to the surface radiance values for each combination of nadir angle and relative azimuth value. The form of the models generated by the program imply a natural organization of the database.

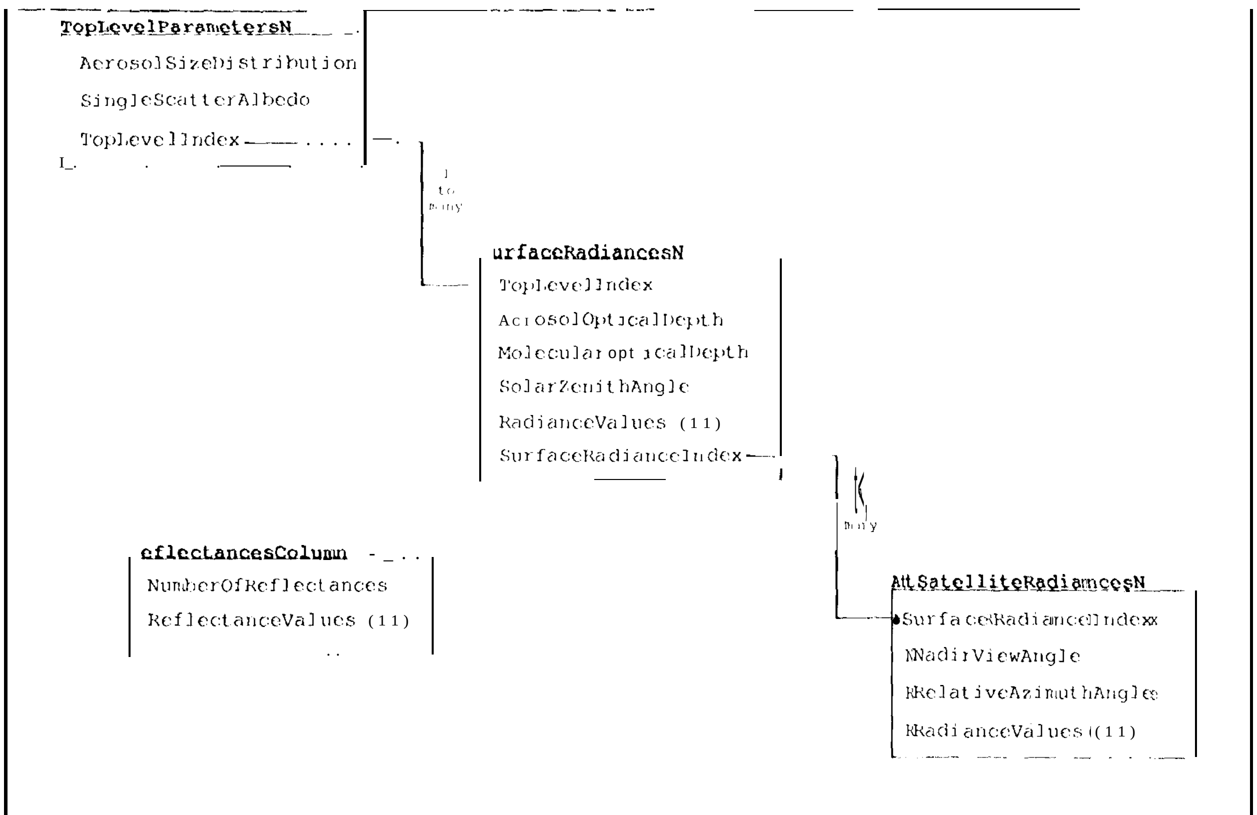

Figure 1. Database schema for 1 ASTER band

4. DATABASE STRUCTURE

The ACLUT consists of a set of models produced by the RTC program described in section 2. An atmospheric model has three columns of 11 real numbers, representing surface reflectance, surface radiance, and at-satellite (or top-of-atmosphere) radiance. The surface reflectance column is the same for every model in the database, so it need appear only once in the database. The surface reflectance column contains the eleven values 0, .1, .2, . . ., 1.0. It is stored in table *ReflectanceColumn*.

The lookup table consists of nine subtables, one for each ASTER band, each with identical structure. Figure 1 illustrates the database schema for a single band. As the illustration shows, for each unique combination of values of aerosol size distribution and single scatter albedo, there are many combinations of aerosol and molecular optical depth and solar zenith angle. For each unique combination of values of all five of these quantities, there is one surface radiance column. Then, for each of these, at-satellite radiance columns vary along with the nadir view and relative azimuth angles.

The choice of values for each of the quantities serving as keys in the database had to be limited by size considerations. A core set of values were identified which would cover the majority of cases encountered in processing the data. The strategy in generating the table was to generate these core cases first, and then generate values to cover more unusual cases later. The set Of values for the core cases is given in Table II

Table 1 I - ***Database key ranges covering the core set of cases***

PARAMETER NAME	SET OF VALUES
Junge Parameter (a.k.a. Aerosol Size Distribution	2.0, 2.25, 2,S, 2.75, 3.0, 3.25, 3.5, 4,0
Single Scatter Albedo	approximately 180 values ranging from 0.08610 .94 by increments Of approximately .004
Aerosol Optical Depth	0.05, 0.1, 0.15, o,?, 0.25, 0.3, 0.35, 0.4, 0.45, 0.S, 0.55, 0.6, 0.65, 0.7, 0,7s, 0.8, 0.85, 0.9, 0.95
Molecular Optical Depth	0.001, 0.031, 0.061, 0.091, 0.121
Solar Zenith Angle.	0.0, 18.194870, '2 S.841 930, 31.788330, 36.869900, 41.409618, 45,573002, 49.4 S8401, 53,130100, 56.632992, 60.000000, 63.256321, 66.42 1822, 69.512680 (these are the arccosine of the numbers 1, .95, .90, ,.., 35)
Nadir View Angle	2.0, 5.0, 8.0, 11.0, 14.0, 17.0, ?0.0
Relative Azimuth Angle	O, 30, 60, 90, 120, 150, 180

For a single band, the numbers of records required to cover the core cases are approximately 290 *TopLevelParameters* records, 383,040 *SurfaceRadiances* records, and 18,768,960 *AtSatelliteRadiances* records. The sixes of the t hı ee t ypes of records are respectivel y 12, 64, and 56 bytes. So the size Of a single subtable fully populated with records covering the core cases is about 1 Gigabyte. The 9 subtables will then lake up about 9 Gigabytes (this doesn't incl ude space foı indexes, procedures, syst em tables, or other tables in the database).

in addit ion to the core set Of cases, the table will be populated with a select ed subset Of cases fı om those coveıed by t he key values shown in Table I11. Exactly which special cases will be covered is not yet determ ined.

Table 1] I - *Database key ranges for extending the* core *set with special cases*

PARAMETER NAME	SET OF VALUES
Junge Parameter (a.k.a. Aerosol Size Distı ibution)	,001, 1.0, 1.5, 4,25, 4.5, 5.0, 6.0
Single Scatter Albedo	similaı to values in the core set Of cases
Aerosol Optical Depth	same as for the core set Of cases
Molecular Optical Depth	same as foı the core set Of cases
Solar Zenith Angle	same as foı the core set Of cases
Nadir View Angle	0, 1.0, 3.0, 4.0, 6.0, 7.0, 9.0, 10.0,12.0,13.0,15.0, 16.0,18.0, 19.0, 22.0, 25.0, 28.0, 30.0
Relative Azimuth Angle	same as for core set Of cases

With one exception, the key values in the database tables are the same values as the inputs given to the RTC program in generating the database (the RTC program is not given a list of values for one of the varying quantities, but rather a minimum, maximum, and delta value for the quantity). The exceptional key is single scatter albedo - it is an output from ıather than an input to the RTC program. That is why the values occur in the database in uneven increments.

Indexes are defined on all of the fields in the tables. These are essential for performance reasons. Each of the three main tables have a *clustered* index; these types of indexes dictate how the table is sorted. For the tables *TopLevelParametersN*, the clustered index is defined on the fields *SingleScatterAlbedo* and *AerosolSizeDistribution*, which means that the tables are sorted first by *SingleScatterAlbedo* and then by *AerosolSizeDistribution*. For the tables *SurfaceRadiancesN*, the clustered index is defined on the fields *TopLevelIndex, AerosolOpticalDepth, MolecularOpticalDepth*, and *SolarZenithAngle*. For the tables *AtSatelliteRadiancesN*, the clustered index is defined on the fields *SurfaceRadianceIndex, NadirViewAngle, RelativeAzimuthAngle*.

In addition to the three main tables for each band shown in Figure 1, there are several auxiliary tables which hold the list of unique values of a given field which actually appear in the main tables. One group of such tables is the group of tables *AerosolSizeDistValuesN*. For band *N*, this table contains the list of distinct values of Aerosol Size Distribution which actually occur in the database table *TopLevelParametersN*. For example, there are only 7 distinct values of Aerosol Size Distribution in table *TopLevelParameters1*, but each value occurs many times in the table. Table *AerosolSizeDistValues1* will contain only 7 entries - one for each distinct value of Aerosol Size Distribution which occurs in table *TopLevelParameters1*. Part of the process of making a query involves finding a closest match to a given value of Aerosol Size Distribution, a value which may not occur in table *TopLevelParameters1*. It is easier to search the small table *AerosolSizeDistValues1* for a match than to search the much larger table *TopLevelParameters1*. In addition to tables *AerosolSizeDistValuesN*, these types of tables exist for the field *SolarZenithAngle* in tables *SurfaceRadiancesN*, and for fields *NadirViewAngle* and *RelativeAzimuthAngle* in tables *AtSatelliteRadiancesN*. Examples of the use of these tables are given in the next section, in which the steps of performing a query are described.

5. QUERIES

Queries are made by calling a *stored procedure* - a SQL (Structured Query Language) program which is compiled and loaded into a database for repeated use. Using a pre-compiled and loaded stored procedure is more efficient than using a complicated SQL script for each query, because the Sybase server is able to do some of the planning work at compile time. A query to the ACLUT is actually carried out by a system of stored procedures. The top-level procedure is named *QueryAtmosModels*. This procedure simply invokes a band-specific sub-procedure based on the ASTER band for which the query is being made. The band-specific procedures have names of the form *OneBandQueryN*, where *N* is the band number. A stored procedure may be invoked interactively at the prompt of a SQL interpreter, or via a subroutine call in a SQL server access library for a specific language. In our case, we use the Sybase C language library *Open Client*.

A query is made with a vector of values for the following parameters:

```
ASTER Band
Aerosol Size Distribution
Single Scatter Albedo
Aerosol Optical Depth
Molecular Optical Depth
Solar Zenith Angle
Nadir View Angle
Relative Azimuth Angle
```

which is passed to the procedure *QueryAtmos Models*. For example, assuming the following vector of parameter values: 1, 2.64, .603, .732, .102, 36.05, 10.5, 77,S, the query would be made with the call

```
QueryAtmosModels 1, 2.64, .603, .732, .102, 36.05, 10.5, 77.5
```

Once invoked, procedure *QueryAtmosModels* performs a series of if-tests on the band number (the first argument), to determine which sub-procedure to invoke. One of these tests is shown in the following excerpt from the SQL code making up procedure *QueryAtmosModels*:

```
if (@ASTERband = 1)
begin
        execute @ReturnValue =
         OneBandQuery1 @ASTERband, @ASdist, @SSalbedo, @AOdepth,
                   @MOdepth, @SolarZen, @NadirView, @RelAzimuth
        return @ReturnValue
end
```

The input parameter values, which are stored in the formal parameter variables *@ASTERband, @ASdist, @SSalbedo, @AOdepth, @MOdepth, @SolarZen, @NadirView*, and *@RelAzimuth*, are passed down to procedure *OneBandQuery1*.

As implied in the previous section, the value of ASTER Band selects a subtable system in which to make the query. For the rest of the parameter values, there may not be an exact match in the database. A search for the closest match to each of the seven floating point parameter values is performed, in the order in which the parameter values are listed. This closest match search is done using the SQL *min* and *abs* functions on the difference between the parameter value and the corresponding field in the table. For the Aerosol Size Distribution parameter, the search for a match is performed in procedure *OneBandQuery1* (still supposing for the sake of example that the value of ASTER Band is 1), and looks similar to this:

```
select @ASdistKey =
        (select max(AerosolSizeDistribValue)
          from AerosolSizeDistValues1
          where abs(AerosolSizeDistribValue - @ASdist) =
            (select min(abs(AerosolSizeDistribValue - @ASdist))
             from AerosolSizeDistValues1))
```

In this SQL excerpt, the variable *@ASdist* contains the query parameter value of Aerosol Size Distribution. Note that the *max* function is needed to break a tie. After the query, the local variable *@ASdistKey* contains the closest match which occurs in table *TopLevelParameters1*. In this example, the value of ASTER Band would have been 1,

Once the closest matching value of Aerosol Size Distribution is found (and stored in the variable @ASdistKey), it can be used to limit the search for the closest match to the parameter value of Single Scatter Albedo, as follows:

```
select @AlbedoKey =
    (select max(SingleScatterAlbedo)
     from TopLevelParameters1
     where AerosolSizeDistribution = @ASdistKey
     and abs(SingleScatterAlbedo - @SSalbedo) =
          (select min(abs(SingleScatterAlbedo - @SSalbedo))
           from TopLevelParameters1
           where AerosolSizeDistribution = @ASdistKey))
```

The next step in doing the query would be to perform similar closest match searches on the next level down - in the table *SurfaceRadiances1* (see Figure 1). First though, the value for the *TopLevelIndex* field in that table has to be found in table TopLevelParameters1. This is done using the closes matches to the other two parameters found by the previous SQL statements shown, as follows:

```
select @TLindex =
     (select TopLevelIndex from TopLevelParameters1
      where AerosolSizeDistribution = @ASdistKey
      and SingleScatterAlbedo = @AlbedoKey)
```

Now the value of *TopLevelIndex*, stored in local variable *@TLindex*, can be used to index table *SurfaceRadiances1* in performing closest fit searches for the parameter values Single Scatter Albedo, Aerosol Optical Depth, and Molecular Optical Depth. These searches are done by this SQL code from the body Of *OneBandQuery1:*

```
select @AOdepthKey =
      (select max( AerosolOpticalDepth)
       from SurfaceRadiances1
       where TopLevelIndex = @TLindex
         and abs(AerosolOpticalDepth - @AOdepth ) =
           (select min(abs(AerosolOpticalDepth - @AOdepth))
            from SurfaceRadiances1
            where TopLevelIndex = @TLindex))

select @MOdepthKey =
      (select max( MolecularOpticalDepth )
       from SurfaceRadiances1
       where TopLevelIndex = @TLindex
         and AerosolOpticalDepth = @AOdepthKey
         and abs(MolecularOpticalDepth - @MOdepth ) =
          (select min(abs(MolecularOpticalDepth - @MOdepth))
                   from SurfaceRadiances1
                   where TopLevelIndex = @TLindex
                     and AerosolOpticalDepth = @AOdepthKey))

select @ZenithKey =
      (select max( SolarZenithAngleValue )
       from SolarZenithAngleValues1
       where abs(SolarZenithAngleValue - @SolarZen) =
         (select min(abs(SolarZenithAngleValue - @SolarZen))
          from SolarZenithAngleValues1))


select @SurfRadIndex =
        (select SurfaceRadianceIndex from SurfaceRadiances1
         where TopLevelIndex = @TLindex
           and AerosolOpticalDepth = @AOdepthKey
           and MolecularOpticalDepth = @MOdepthKey
           and SolarZenithAngle = @ZenithKey)
```

In this example, the auxiliary table *SolarZenithAngleValues1* is searched for a closest match for the input parameter *@SolarZen,* which contains the Solar Zenith Angle value in the query parameter vector. The use of this table depends on the assumption that every value of *SolarZenithAngle* which appears in table *SurfaceRadiances1* appears with every value of each of *TopLevelIndex, AerosolOpticalDepth,* and *MolecularOpticalDepth* which occurs in the table.

Having found the correct value for *SurfaceRadianceIndex*, the surface radiances can now easily be found:

```
select RadianceValues
from SurfaceRadiances1
```

```
        where SurfaceRadianceIndex = @SurfRadIndex
```

and the last level of the query can be done in table *AtSatelliteRadiances1*:

```
select @NadirViewKey =
        (select max(NadirViewValue)
         from NadirViewValues1
         where abs(NadirViewValue - @NadirView) =
                  (select min(abs(NadirViewValue - @NadirView ))
                   from NadirViewValues1))

select @AzimuthKey =
        (select max(RelativeAzimuthValue) from RelativeAzimuthValues1
         where abs(RelativeAzimuthValue - @RelAzimuth) =
              (select min(abs(RelativeAzimuthValue - @RelAzimuth ))
               from RelativeAzimuthValues1))

select RadianceValues from AtSatelliteRadiances1
         where SurfaceRadianceIndex = @SurfRadIndex
           and NadirViewAngle = @NadirViewKey
           and RelativeAzimuthAngle = @AzimuthKey
```

Again, the use of auxiliary tables to find the closest-matching values for Nadir View Angle and Azimuth Angle rests on the assumption that every value of each of those quantities occur with every value of the other field and of the *SurfaceRadianceIndex* field in table *AtSatelliteRadiances1*.

The *RadianceValues* fields in both tables *SurfaceRadiancesN* and *AtSatelliteRadiancesN* are of type *binary(44)* - which in Sybase's implementation of SQL is simply a string of 44 bits, which is how many bits are needed to store 11 floating point numbers. So, the output from the invocation of procedure *QueryAtmosModels* shown above is:

```
0x000000003c81831 93d02602d3d44e50c3d842d8c3da65bea3dc902dc3dec22683e07e2823e1a02753e2c56d6
0x3c2454de3caa4fca3d01cd603d2f10233d5cfaad3d85c1403d9d590c3db544bb3dcd9e843de631f93dff2e49
```

The translation of these bit-strings into floating-point numbers is done in software, which is described in the.next section. The procedure *QueryAtmosModels* actually returns more than just the two bit strings - it also returns a boolean flag which indicates whether the query produced results, the values of *TopLevelIndex* and *SurfaceRadianceIndex* found, and the matching value found for each of the eight query parameter values.

7. CUSTOM DATABASE ACCESS SOFTWARE

During the initial planning and design phase for the implementation of the ACLUT and access to it by the atmospheric correction software, it was not known how complex the software that communicated with the database server would need to be. There might have been a need for storing part of the table in memory for performance reasons, or a need to do interpolation on the query results.

In addition, the, Sybase Open Client library routines can be cumbersome to use. For example, to issue a simple SQL query to the server, retrieve the, results for one row and place the results in program variables requires a minimum Of five calls to routines in the Sybase library. For these reasons, a library Of database access routines was developed, bawd on Open Client routines. That way later enhancements could be encapsulated in the library code, and not affect the larger atmospheric correction software itself. The library is called the Database Interface library (DB_IF). Table IV gives a list of the routines in the library and a brief description of the function of each

Table IV - *Routines in the ASTER database interface library*

ROUTINE NAME	DESCRIPTION
db_if_ClientLibCallback	Invoked when the Sybase client library issues a callback; prints information about the, error or warning.
db_if_CloseHandle	Closes connection to database, releases memory associated with management of connection.
db_if_DoQuery	Performs query given a list Of parameter vectors.
db_if_GetSurfaceReflectances	Retrieves the surface reflectance column for the table.
db_if_OpenConnection	Opens a generic connection to the database.
db_if_OpenTable	Opens a connection to the database and also selects the internal database appropriate for the requested version of the table. Retrieves the reflectance column.
db_if_SendCommand	Sends a command - usually a SQL statement - to the database server.
db_if_ServerCallback	Invoked if the Sybase server issues a callback; prints information abou the error or warning.

The routine of most interest is *db_if_DoQuery*. The following excerpt from that routine's C code. shows the interface.

```
db_if_status
db_if_DoQuery(db_if_handle Handle,
              db_if_QueryKey * KeyList,
              int NumKeys,
              db_if_QueryResult * ResultList)
{



}
```

The variable *Handle* is a data structure which contains all control and status information about the open connection to the data.server. The value of *Handle* would have been returned previously by one of the routines *db_if_OpenTable* or *db_if_OpenConnection*. The variable *KeyList* is an array of query parameter vectors, and variable *NumKeys* contains the number of items in the array. The output argument *ResultList* is an array of elements capable of containing a result from a query (including the two columns of radiance values).

Each item in the array KeyList is a value with a structure described in C by the following definition:

```
typedef struct db_if_QueryKey {
    int         BandNumber;
    float       AerosolSizeDist,
                AerosolSingleScatterAlbedo,
                AerosolOpticalDepth,
                MolecularOpticalDepth,
                SolarZenithAngle,
                NadirViewAngle,
                RelativeAzimuthAngle;
}  db_if_QueryKey;
```

This excerpt defines a type named *db_if_QueryKey*, which is the C type of each element in array *KeyList*.

When *db_if_DoQuery* runs, it loops through the list of queries to be made. For each query, it formats the values in the query parameter vector into a string containing a SQL call to the procedure *QueryAtmosModels*. It then sends this query to the server, and retrieves the results into the appropriate element in the array *ResultList*. At this stage the routine must also decode the two bit strings into the floating-point numbers of which the surface radiance and at-satellite radiance columns consist. The structure of the results of one query is defined by the following C statement:

```
typedef struct db_if_QueryResult {
    int             HitFlag;
    float           SurfaceRadiances[11],
                    TopOfAtmosRadiances[11];
    int             TopLevelIndex,
                    SurfaceRadianceIndex;
    db_if_QueryKey  KeyMatch;
}                   db_if_QueryResult;
```

Each element in the array *ResultList* has this structure. The value for each field in the structure is returned by the stored procedure *QueryAtmosModels*. Note that the structure contains a substructure of type db_if_QueryKey - this is for storing the matching values for each value in the query parameter vector. This information is used for diagnostics and testing, and, it turns out, for optimization of the query process.

8. QUERY PERFORMANCE AND OPTIMIZATION

Doing a single query takes approximately 5 seconds. Current plans call for making approximately 150 queries per band per scene, which translates to 1350 queries since there are nine bands. The time to do this many queries would translate into approximately two hours of wall clock time. This is not desirable, so an attempt has been made. to optimize the query process.

There is no way to make the database query itself go much faster than it does. We could perhaps squeeze out a few seconds by making the table smaller, but that would not make enough of a difference to sacrifice the accuracy that would be lost or to accept the added complexity of populating the table. The strategy used instead has been to cache the results of queries and be able to reuse them when appropriate.

As shown in Figure 1, the database is organized into a three-level table structure, and querying takes place in three groups of discreet steps, one group on each level. At each level information is frond that is used on the query steps in the next level down. For example, first the top level is queried to find a value for *TopLevelIndex*. This value is then used to index the table on the second level down. If a query were made in which the correct value for *TopLevelIndex* were known before hand, the top-level query steps could be skipped, and the query begin on the second level down. Similarly, if the! appropriate values of *TopLevelIndex* and *SurfaceRadianceIndex* were known before hand, the query could begin on the bottom level with table *AtSatelliteRadiancesN*. This latter type of query takes less than one second

In many of the scenes to be processed, we conjecture that only the instrument view angle and nadir angle will change in the, course Of a scene. If a scene is of uniform altitude and atmospheric conditions, the optical depth values, aerosol size distribution, and single scatter albedo won't change over the scene. In processing a scene like that, all but the, first query could proceed directly to the bottom level, since the values of *TopLevelIndex* and *SurfaceRadianceIndex* would not change during the scene. Doing 1350 queries for that scene would take less than 1400 seconds.

Even in scenes with more atmospheric variety, it is likely that the single scatter albedo and aerosol size distribution will change less than the molecular and optics] depth values. In that case, many of the queries could begin with the second-le.vcl, because the, albedo and aerosol size distribution would not have changed from the last query (or another recently-done query), and so *TopLevelIndex* would be known. A query of this type takes about 2 or 3 seconds, which is significantly faster than a query done with no fore knowledge.

This type of query optimization is implemented in the routine *db_if_DoQuery*. That routine sets up and maintains a query cache as a queue of queries and their results. Each time the routine is **called** to do a query, it first searches the query cache looking for a successful query that matches to some degree the current query. The.re are three degrees of matching: a perfect retch, in which all seven of the floating-point quantities in the query parameter vector match; a 2-level match, in which the *AerosolSizeDistribution* and *SingleScatterAlbedo*, *AerosolOpticalDepth*, and *MolecularOpticalDepth* match; and a 1 -level match, in which only the *AerosolSizeDistribution* and *SingleScatterAlbedo* values match. If a perfect match is found with some query in the cache, the query results of that query arc simply returned to the caller without consulting the database at all (in this case a "query" takes only micro-seconds). If a 1-level match is found, the query sent to the database includes the value Of *TopLevelIndex*, so that the query can begin on the second level. If a 2-level match is found, the software sends a query which includes values for both the *TopLevelIndex* and the *SurfaceRadianceIndex*, thus allowing the actual query to begin on the bottom-level.

At this time we don't know how effective this optimization will be, since we have only limited simulated atmospheric profile datasets.

9. DATABASE GENERATION PROCESS

The Sybase database server is being used to manage the table generation process as well as to store the table's data. The sets of input parameter values for all runs of the RTC have been enumerated and stored in a database table called *TransferCodeRuns*. In addition to all values needed for inputs to the RTC, this table also has status and priority fields which can be used to prioritize and manage the runs. We've developed a program, called *db_gen*, which queries this table based on status and priority to select an RTC input parameter set. It then runs the RTC, converts the RTC's outputs into binary data, and inserts the data into the database.

Db_gen has been running since the winter 1995 on three Spate 2(I Sun workstations, two Of them dual processor machines, at the University of Arizona, The Sybase dataserver is running on a Spare 10 workstation at Jet Propulsion Laboratory. The program makes a connection from the host machines at U. of A. across the. Internet to the database Server at JPL. Occasionally delays through the connection cause errors in the data insertion process but for the most part the process is quite robust.

To date, 1087 RTC runs have been successfully made and inserted into the database. There are approximately 3000 more runs to do to complete the table.

9. ACKNOWLEDGMENTS

This work was performed as part of ongoing research at California Institute of TeclmOIC)gy/Jet Propulsion Laboratory. The authors wish to express our appreciation to NASA for supporting this work under contracts NAS71260 (for JPL phases of the work) and NAS5-31717 (for U. of Arizona participation).

10. REFERENCES

1. Kahle, A., "U.S. Science Team Leader Experiment Implementation Plan", Jet Propulsion Laboratory Document #/ 1)-11 2000, 3/19/1994

2. Herman, B. M., and S.R. Browning, "A numerical solution to the equation of radiative transfer," J. Atmos. Sci., 22, (1 965), 559-566.

3. Iqbal, M., An Introduction to Solar Radiation, New York: Academic Press, 1983, 380;381

4. Slater, P., Digger, S., and Thome, K., Algorithm Theoretical Basis Document for ASTER Surface Radiance. and Reflectance, Remote Sensing Group/Optical Sciences Center, University of Arizona, Jan, 21, 1995

5. Thome, K . J., "Proposed atmospheric correction for the solar<AD>reflective bands of the Advanced Spaceborne Thermal Emission and Reflection Radiometer," International Geoscience and Remote Seining Symposium, Pasadena, pp. 202-204, 1994.

6. "MODTRAN: A Moderate Resolution Model for LOWTRAN 7", Berk, A., Bernstein, L., Robertson, D., GL-TR-89-0122, Air Force Geophysics Laboratory, 1989.